APPENDIX L

Appendix L will be understood and appreciated from the following detailed description, taken in conjuction with the drawings in which:

Fig. 75 is a polyline object illustrating the deviation parameter of a process for straight line approximation of connected components composed of CELs, constructed and operative in accordance with a preferred embodiment of the present invention;

Fig. 76 is a polyline object illustrating the deflection parameter of a process for straight line approximation of connected components composed of CELs, constructed and operative in accordance with a preferred embodiment of the present invention;

Fig. 77 is a graph illustrating the small_cel length parameter of a process for straight line approximation of connected components composed of CELs, constructed and operative in accordance with a preferred embodiment of the present invention;

Fig. 78 is a polyline object illustrating a 'quality assurance' mechanism in a process for straight line approximation of connected components composed of CELs, constructed and operative in accordance with a preferred embodiment of the present invention;

Fig. 79 is a polyline object illustrating an advanced 'quality assurance' mechanism in a process for straight line approximation of connected components composed of CELs, constructed and operative in accordance with a preferred embodiment of the present invention;

Fig. 80 is a graph illustrating the results of a performance test in a process for straight line approximation of connected components composed of CELs, constructed and operative in accordance with a preferred embodiment of the present invention;

Fig. 81 is a graph derived from the graph of Fig. 80 by examining the situation where the compression ratio (as a function of deflection angle) becomes constant for every vectorization style; and

Fig. 82 is a graph derived from the graph of Fig. 80 by examining the deflection angle where the compression ratio (as a function of deflection angle) becomes constant for every vectorization style, and the deflection angle that produces minimum time vectorization.

INTRODUCTION

A key problem in computer vision is the extraction of meaningful features from images. A popular approach is based on edge data which, though useful itself, is better represented in a more manageable form. The type of description can be application dependent but is usually based on a combination of straight line approximations and if possible approximations with higher order curves such as circular arcs, conic sections, splines, and curvature primitives. Straight lines are standard inputs to model matching systems, but higher order features are becoming more popular since they permit more constraints on model identification.

In this Appendix, a method is described for straight line approximation of connected components composed of CELCELs (lists of edge elements generated using edge detection method producing single pixel contour elements [CELCELs] followed by edge linking). Among the properties of the method we find:

* *Efficiency* - Good representation of the data up to a desired tolerance along with substantial data reduction, so that higher order geometric representations can be easily fitted to the data. All this is done with relatively low time consumption.

* *Low dependence on parameters* - The method has a small number of parameters which affect its operation, some affect the time and memory consumption, and another independent one affects the output data quality (preservation of the shape).

OVERVIEW OF THE METHOD

General

At present the method is designed to work on highly sampled contours of a shape, constructed of single pixel contour elements (CELs) connected into chains. All the preparation processes are done by a series of binarization and raster to vector methods. In general we can look at the problem as getting a contour of a shape and producing another (compressed, more efficient) description as output.

The main goal of the method is to produce a good and efficient representation of the input data for later processing (reference for change detection ,shape recognition etc.). A "good representation" can be defined as a representation preserving the original shape up to a desired tolerance and removing redundant data as much as possible.

In general, the operation of the method is based on running on the input contour and searching for significant points or vertices for the description of the shape. The significance of a vertex is determined using angle difference criteria of an examined

CEL of the input contour with respect to its environment. When a significant vertex is found the method starts a strong examination of "preservation of the original shape" and improves the vectorization, by adding more vertices, to achieve a desired deviation tolerance. The method has also a simple noise reduction criterion that improves data reduction by ignoring small contour elements that add little information to the description of the shape. The following sections describe in detail the method's input, output, parameters and operation mechanisms.

Input and output

The input to the method is a connected components object which is a collection of highly sampled contours of the shapes in the input image. As mentioned earlier the input connected components object is a list of edge elements generated using edge detection method (producing single pixel contour elements or CELs) followed by edge linking method. The connected components are the first form of vector representation of edges of the input image in the system.

The output of the method is a polylines object which is a collection of contours of shapes of the input image. The polylines object is the basis for all shape modelling and shape comparison in the system.

Parameters

The method has three input parameters that control its operation:

* The most important parameter that controls the output data quality is the deviation (in pixels length units) tolerance of the output contour from the input contour. This parameter influences the smoothing of the input contour and also the data compression ratio.

Reference is now made to Figure 75 which aids in the understanding of one preferred embodiment of the present invention.

* The next parameter is the deflection (in degrees units) threshold, which is used to determine the importance of the examined vertex for the description of the shape. The main influence of this parameter is in the determination of the speed and memory consumption during the process. All the aspects of this parameter are discussed below in the performance test and results section.

Reference is now made to Figure 76 which aids in the understanding of one preferred embodiment of the present invention.

* The third parameter which is used to reduce noise is the small_cel length (in pixels length units) which is used to filter out short contour elements that have a highly unstable angular nature with respect to their neighbourhood. The small contribution of the small cels to the description of the shape allows us to automatically discard them ,as non important vertices ,from further analysis.

Reference is now made to Figure 77 which aids in the understanding of one preferred embodiment of the present invention.

Adaptors for the function parameters are used to improve efficiency.

1. The deflection angle is converted from degrees to radians
in order to save the transformation in the angle calculation
function of the examined segment.

2. The deviation and small_cel sizes are converted to square of
the size to save the sqrt() function all the time a distance
has to be calculated.

```
rdeflection = deflection * (PI / 180);
minus_rdeflection = 2 * PI - rdeflection;
sq_small_cel = small_cel * small_cel;
sq_deviation = deviation * deviation;
```

Operation

The main iteration of the method is performed on the connected components object where every chain of CELs is processed sequentially. The main loop first declares a new polyline in the output polylines data object and then executes the Vectorize_chain function, which prepares the desired result in the output data object. At the end of vectorization function the new polyline is closed and marked with some additional system data for further use.

Execute code

```
// run over all chains and vectorize each one of them
 for ( unsigned int i = 0; i < c_components->size(); ++i )
 {
  data->BeginPolyline();
  VectorizeChain( data, (c_components)[i],
rdeflection,
```

```
sq_small_cel,
sq_deviation);
 data->EndPolyline((c_components)[i].IsClosed(),
(c_components)[i].Id(),
(c_components)[i].MtrlType());
 }
```

Vectorize chain

The Vectorize_chain function receives as parameters, pointers to the input and output objects and the three operational parameters described above. The function simply runs on the input chain (of CELs) and looks for "significant" vertices for the description of the shape. If such an important vertex is found, it is added to the output object.

It is appreciated that some exceptional cases exist, e.g. an empty chain of cels that can not be processed so the function exits. Also if ,for some reason, the chain includes only one vertex, there is no need for processing so the function inserts the first vertex into the output object and exits.

Another convention that should be mentioned here concerns the geometric properties of vertices and segments of connected components and polylines in the system:

* Angle(i) is the angle of i'th segment (first endpoint of CEL or line) pol[i]-->pol[i+1]

* Length(i) is the length of i'th segment (first endpoint of CEL or line) pol[i]-->pol[i+1]

* Sq_length(i) is the Squared length of i'th segment.

After the first special cases checking the function initialises the angle variable to the angle of the first segment, and resets an index for further use. At this time we are ready to start running along the chain and find "important" vertices. For each vertex we have the segment angle and length and the angle difference to the entrance angle of the examined group of segments (a group of segments is defined between every two "important" vertices in the output polyline).

If the examined CEL's angle_difference is larger then the deflection threshold it is considered an important vertex for the description of the shape.

When examining this "vertex importance" special cases are considered for segments with north azimuth that jumps from small angles (around 0 degrees) to large angles (around 360 degrees). Additional part of the question includes the "small cels" filter which bypasses the cases where a segment (a very short CEL) is accompanied by very large deflection angles on both his sides that crosses the deflection threshold and adds vertices which do not add important information to the description of the shape and substantially reduces the compression ratio.

If it was found that the examined vertex is "important for the description of the shape" it is typically inserted it into the output data structure. As a result of this insertion a new line will be created between the previously inserted and the newly found vertices. For some reasons it could happen that the newly created segment substantially deviates from the original contour. This incident is common along low curvature and smooth contours. In order to avoid the loss of information caused by these cases, the proposed vectorization are typically rechecked for deviations with respect to a given tolerance. This checking is done by the Improve_segment function before adding the new "important" vertex into the output object.

In general, the Improve_segment function checks that the proposed segment does not deviate from the original contour more then the permitted distance, and adds additional vertices if used. After improvement is done the new proposed vertex is inserted into the output vector and some more data is updated to continue the iteration. After the whole chain is processed we typically conclude by inserting the last vertex, of course with improvement of segment before insertion. The detailed operation of the improvement function is described below.

Vectorize_chain code

```
{
unsigned improve_sample = 2;  // index
if ( ipol.size() == 0 ) return; // connected component contains no vertices

data.push_back( ipol[0] );
if ( ipol.size() == 1 ) return; // connected component contains only 1 vertex
double  entrance = ipol.Angle(0);
unsigned startind = 0;
```

```
for (unsigned i = 1; i < ipol.size()-1; ++i )
{
 // Get azimuth and square length of the tested segment
 // and calculate angle difference.
 double angle   = ipol.Angle( i );
 double diffangle = fabs(angle-entrance );
 double sq_length = ipol.Sq_length( i );
 // Check significance of current vertex
if ((diffangle > rdeflection) andand (diffangle < minus_rdeflection) andand
(sq_length > sq_small_cel))
 {
  // About to add a vertex
  // Improve the new segment sapling
Improve_segment(ipol, startind, i, improve_sample,
  sq_deviation, data, cel_touch);
  data.push_back( ipol[i] );
  entrance = angle;
  startind = i;
 }
}
// About to add a last vertex
  // Improve the new segment sampling
Improve_segment(ipol, startind, ipol.size()-1, improve_sample,
sq_deviation, data, cel_touch);
data.push_back( ipol[ipol.size()-1] );

return;
}
```

Improve segment

The decision rules used to define "important vertices" along the chain has some weakness in detecting very low deflections of the contour, a problem that could result in a low quality description of the shape. As a result of this, and other possible

deficiencies, some mechanism for quality assurance of the vectorized result is typically used. The mechanism used is a very simple examination of the difference between the proposed vectorized resulting segment and the input chain of cels which it is meant to replace.

The Improve Segment function works on a predefined segment of the input connected chain of cels (starting and ending indices of the desired cels in the chain, defined by the upper level Vectorize Chain function). The examination is performed along the whole input segment and the vertex with maximum distance is pointed out. If the distance is larger then the allowed parameter then the most distant vertex is also added to the output polyline.

The measurement of the distance between the examined cel the proposed output segment is based on the measurement of the area of the triangle between the two proposed end points and the point on the segment of cels which is bounded by these points, as described in Equation 2.

The desired length is equal to (2*Area of the triangle)/(length of the base) where the length of the base is the distance between points p0 and p1 (Fig. 78). Since the base length is constant during the whole improvement iteration we can first calculate the base length term and save some calculations. Since the improvement iteration adds another touch to every cel in the input chain there is a big motivation to reduce the number of measurements. In the improvement iteration on the chain of cels a sampling parameter is used which skips a desired number of CEL distance measurements. The assumption behind this sampling parameter is that there is a relatively smooth contour of CELs between the starting and ending points that for some reason was not detected by the vectorize chain function and any high curvature features are not lost so a sparser quality checking is allowed.

It was mentioned hereinabove that the Improve segment function is preferably called before insertion of every new vertex into the output polygon in order to keep the right order of vertices of the output shape. Since the improve segment function inserts vertices itself preferably are called recursively call it before inserting additional vertices and also call it after the insertion in order to refine the output polygon according to the desired quality parameter.

Improve_segment code

```
{
 if ((endind-startind) < improve_sample) return;
 Dpoint2 p0 = ipol[startind], p1 = ipol[endind];
 double area2, sq_dist, maxdist = 0;
 double sq_seg_length = p0.Sq_dist( p1 ); // base length
 unsigned maxind = 0;
for (unsigned j = startind; j < endind; j+=improve_sample)
 {
  area2  = (p1.y - ipol[j].y)*(p0.x - ipol[j].x) -
      (p0.y - ipol[j].y)*(p1.x - ipol[j].x);
  sq_dist = area2*area2/sq_seg_length;
if (sq_dist > maxdist)
  {
   maxdist = sq_dist;
   maxind = j;
  }
  cel_touch++;
 }
 if (maxdist > sq_deviation)
 {
  Improve_segment(ipol, startind, maxind, improve_sample,
  sq_deviation, data, cel_touch);
  data.push_back( ipol[maxind] );
  Improve_segment(ipol, maxind,  endind, improve_sample,
  sq_deviation, data, cel_touch);
 }
 return;
}
```

3. PERFORMANCE TEST AND RESULTS

When trying to estimate the performance of the method we preferably define the relevant quality parameters which are:

* preservation of the shape

* compression ratio of redundant data

* time consumption in the specific scenario

Some quality parameters are very rigid and are dictated by the user application, for example the preservation of the shape, while other parameters influence the operation of the method but not the output result that the user receives.

The compression of redundant data is the ratio between the number of contour elements in the input connected components data object and the output number of vertices in the output polylines object while the time consumption was measured using a timer object.

It is appreciated that the two last quality parameters can influence the operation of the whole system on conditions where time is a very important parameter that should be satisfied on some operational specifications, For example an unwise selection of the deflection parameter in learn scan can result with a low compression ratio which can degrade the performance of test functions in inspect scan.

For the performance test of the method a sample panel is used containing about 93000 contour elements (CELs) that are connected to contour chains using a separate connected components method. For this panel, the influence of the various input parameters on the output quality is tested.

In every test phase it is assumed that the preservation of the shape parameter is rigidly dictated by the user (this quality parameter is directly controlled by the deviation input parameter described above) and the deflection parameter is varied while testing its influence on the compression ratio and the time consumption of the method execution. During the whole test the small_cels length parameter is typically fixed to 0.3 pixel and the improvement resampling index parameter is typically fixed to 2 (every second CEL is examined in the improvement iteration). The whole set of deflection variations is tested using three deviation conditions: "tight" (0.1 pixel deviation) "medium" (0.3 pixel deviation) and "loose" (0.6 pixel deviation). The whole test is carried out on a Intel Pentium pro (dual processor) computer which was completely devoted for the analysis with out any users or processes.

The test results are displayed on Figure 80. On the results we first notice the inverse relation between the compression ratio and the time consumption as changing

from tight to loose vectorization (tighter vectorization consumes more time and delivers lower compression ratio). We can also see the logarithmic nature of the compression ratio as a function of the deflection parameter and that every vectorization style has an a constant compression level after some deflection limit, which implies that the final compression ratio is mainly controlled by the deviation parameter.

Observation of the time consumption data reveals the existence of a minimum value of execution time. Examining the code it can be seen that the time consumption is directly related to the number of CELs in the Connected components structure because the Vectorize chain function touches every CEL once (constant time consumption) and the Improve segment function touches the CELs some more times according to the necessary improvements. The overhead of the Improve segment function is added in any case no matter if the improvement is found necessary or not during a specific iteration.

In the performance tests done with the function, an improvement skip parameter of 2 is typically used which means that the number of touches added by the Improve segment function is 0.5 of the number of CELs in the case where no improvement is needed. This parameter remains constant during the whole operation test and its influence was not tested.

After all these explanations it can be concluded that the minimal number of CEL touches (of any kind) done by the function is 1.5 times the number of CELs in the input Connected components object one touch by the Vectorize chain function and at least 0.5 touch (in the usually used configuration) by the Improve segment function.

The "real" number of CEL touches was measured by counting the overall number of touches performed by the Improve segment function and is found to vary significantly as a function of the deflection parameter. For example if using the "0.3 pixel deviation" vectorizer and setting the deflection parameter to 10 degrees the result shows that the number of touches performed by Improve segment is 0.74 (resulting with 1.74 touches) of the number of CELs in the input. The excess touches (above ratio of 0.5) are caused because of the recursive calls to the Improve segment function.

Changing the deflection parameter to 20 degrees resulted with an overall CEL touches of 2.84 (1 + 1.84) of the CELs number in the input. A deflection parameter of

30 degrees for example causes an overall ratio of 3.34 because of many recursive improvement operations.

The overhead value of 0.508 is found when used a value of 7 degrees is used a the deflection parameter which means that almost no recursive operations were used in this iteration.

When the deflection parameter value is reduced to 3 degrees the resulting overhead is found to be 0.32 which means that not all the CELs are inspected by the improve segment function. This result is found because the improvement is requested for segments with length smaller then 2 CELs (see the first line in the Improve_segment code).

It can thus be concluded that the overhead touches result, which is controlled by the deflection parameter, strongly affects the time consumption of the function.

It is appreciated that the whole performance analysis was performed using a fixed small cel filter of 0.3 pixel. The influence of this parameter on the operation of the method is considered minor on the basis of the accumulated operational experience with the system.

CONCLUSIONS

When coming to select parameters for the operation of the method, if it is assumed that the deviation parameter (or the requested preservation of the shape) is dictated by the application, it can be concluded that the deflection parameter is the main control on the compression ratio and the time consumption of the method.

In the currently common use of the method (preparation of reference vector data for shape change detection) the main goal is to supply the maximum compressed result without unnecessary usage of time. This is especially true in the preparation of reference data of areas which can not be characterised during learn scan, forcing the necessity to perform vectorization during inspection scan.

In the following graphs it can be concluded that the operational recommendation for parameters setting of the method. The first graph shows the maximum compression ratio as a function of the deviation tolerance parameter. This is the first piece of information that has to be considered when coming to design vectorization for a specific application.

From the graphs in figure 80 it is obvious that requesting a tighter vectorization (better preservation of the shape) will cost extra time and reduce compression ratio, so preferably figures out the optimal shape preservation for a given application to meet time consumption constraints.

Figure 81, which is derived from figure 80, is a graph created by examining the situation where the compression ratio (as a function of deflection angle) becomes constant for every vectorization style (or for a specific deviation parameter setting). The nearly linear nature of this function in the examined range (between 0.1 to 0.6 pixel deviation) is evident from the graph. The performance of the method was not tested beyond this range because in the 0.1 pixel deviation tolerance condition was found to produce a highly redundant result for our application and the 0.6 pixel condition produced poor and useless "preservation of the shape" results.

In accordance with a preferred embodiment of the present invention, a deviation value of 0.3 pixel which is used (with typical defect tolerances of several applications). If for some applications the proposed vectorization is not adequate for the preservation of the original shapes, the parameter may be changed, remembering that the maximum compression ratio produced by the vectorizer will change accordingly.

The next piece of information that has to be considered is the deflection angle that has to be used for every deviation setting that will result in an optimal vectorization (best compression with no unnecessary time consumption).

Figure 82, which is a graph derived from figure 80 was created by examining the actual deflection angle where the compression ratio (as a function of deflection angle) becomes constant for every vectorization style (or for a specific deviation parameter setting). The nearly linear nature of this function is also evident from the graph.

In addition to the optimal compression recommendation I showed in the graph a deflection angle that will produce minimum time vectorization as derived from the local minima in the time consumption graphs for every deviation parameter setting.

Even though there is a "minimum time consumption" situation for every vectorization style (or deviation tolerance) it is typically not recommended to perform such "minimum time consumption" vectorizations ,even with very tight time constraints scenarios, because of the poor output compression ratios, that may have to be paid later in the operation of further methods (such as shape change detection).

It is appreciated that the whole performance analysis was performed using a fixed small cel filter of 0.3 pixel and an improve resampling step of 2. The influence of these parameters on the operation of the method is considered minor on the basis of accumulated operational experience with the system.

It is anticipated that automatic tools may be able to automatically produce the necessary data for calibration of the parameters for optimal vectorization as dictated for every application the system will meet. In general the tool has to produce the two dimensional functions of compression ratio and time consumption as a function of the deviation and the deflection parameters.

EQUATION 2

$$2 * area = \begin{vmatrix} p0.x & p0.y \\ p1.x & p1.y \\ pi.x & pi.y \end{vmatrix}$$

(11 operations: 6 multiply, 3 subtract, 2 add)

which can be reduced to:

2 * Area = (p1.y - pi.y) * (p0.x - pi.x) - (p0.y - pi.y) * (p1.x - pi.x)